Efficient Similarity Search across Top-k Lists under the Kendall's Tau Distance*

Koninika Pal
TU Kaiserslautern
Kaiserslautern, Germany
pal@cs.uni-kl.de

Sebastian Michel
TU Kaiserslautern
Kaiserslautern, Germany
smichel@cs.uni-kl.de

ABSTRACT

We consider the problem of similarity search in a set of top-k lists under the generalized Kendall's Tau distance. This distance describes how related two rankings are in terms of discordantly ordered items. We consider pair- and triplets-based indices to counter the shortcomings of naive inverted indices and derive efficient query schemes by relating the proposed index structures to the concept of locality sensitive hashing (LSH). Specifically, we devise four different LSH schemes for Kendall's Tau using two generic hash families over individual elements or pairs of them. We show that each of these functions has the desired property of being locality sensitive. Further, we discuss the selection of hash functions for the proposed LSH schemes for a given query ranking, called query-driven LSH and derive bounds for the required number of hash functions to use in order to achieve a predefined recall goal. Experimental results, using two real-world datasets, show that the devised methods outperform the SimJoin method—the state of the art method to query for similar sets—and are far superior to a plain inverted-index–based approach.

1. INTRODUCTION

Ranked lists, specifically top-k rankings, are ubiquitous concepts that order items according to measurable criteria or crowdsourced user votes. The list of the tallest buildings in the world or the list of the World's wealthiest men are prominent examples, while the use of such ranked lists is virtually unlimited—we can see them in all kind of domains. In this work, we address the problem of performing efficient similarity search over top-k rankings; for a user-provided query ranking and similarity threshold. Finding similar rankings, in the sense that they report on a similar set of entities in roughly the same order, carries valuable analytical insights. The scope of similarity search over ranked lists is not only bound to information systems, but it is also very common in other fields like behavioral studies in social sciences or studies on scientific data in experimental science.

There exist several prominent distance functions, such as Kendall's Tau, Spearman's footrule distance, NDGC [18], the rank distance [5], and ERR [6]. With respect to simplicity, generalization, richness, and basic properties of these metrics, the two predominant distance metrics in literature are Kendall's Tau distance and Spearman's foot rule distance, discussed in [20]. Both are originally defined over pairs of rankings that capture the same (full) domain of elements. For incomplete rankings of size k, also called top-k lists, Fagin et al. [11] describe generalizations of Kendall's Tau and Spearman's foot rule distance to handle incompleteness. However, Fagin et al. [11] show that the generalized Kendall's Tau distance violates the triangle inequality property, which eliminates the scope of using metric-space index structures, like the M-tree [7] and, thus, render the similarity search over top-k list with Kendall's Tau challenging.

In this paper, we address the r-near neighbor (r-NN) problem where the generalized Kendall's Tau distance is used as the distance measure. There are few spatial indexing methods available like the R-Tree [14] and the K-D Tree [4] that are able to produce exact results for the NN-search but are hardly applicable for high-dimensional data. Weber et al. [29] show that the efficiency of NN-search using such structures becomes even worse than a brute-force linear-scan if the dimensions become higher than ten.

Harnessing the observation that at least one element should be contained in two given rankings in order to have a reasonable minimum similarity between them, one classical solution to r-NN search is the application of inverted indices. Such indices are very efficient in answering set-containment queries [15]. We will compare the proposed indices to plain inverted indices, a full linear scan, and the state-of-the-art approach [28] for querying for similar sets.

In this work, we employ three different inverted index structures considering the characteristics of Kendall's Tau distance; using pairs or triplets of items as the indexing granularity. When analyzing the query processing performance in such approaches, we observe that it is often *sufficient to query the index with a small subset of the query's items*, hence, leading to an approximate but very efficient query processing. To understand and theoretically derive the expected performance and accuracy, we relate these concepts to locality sensitive hashing (LSH). LSH has proven to be a very efficient method for NN search [1] for high-dimensional data. In literature, LSH was proposed for various different metric distances such as l_1, Euclidean, or Hamming distance [8, 17]. Although Kendall's Tau is *not a metric distance function*, we observe a relationship between Kendall's Tau and the Hamming distance and also with the Jaccard distance, which motivated us to work on LSH hash families

*This work has been supported by the German Research Foundation (DFG) under grant MI 1794/1-1.

SSDBM '16, July 18 - 20, 2016, Budapest, Hungary

ISBN 978-1-4503-4215-5/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2949689.2949709

for the Kendall's Tau distance.

Considering a uniform distribution of data, LSH methods commonly select hash functions randomly from a hash function family to map the query data into the hash tables to find similar objects. However, real world data is prone to having a non-uniform distribution and it is difficult to capture characteristics of the queries without any prior knowledge or restrictive assumptions. Hence, in this work, we focus on a query-specific selection of hash functions to map the query to the hash table, called query-driven LSH, in order to render LSH more efficient for similarity search—without prior knowledge of the query data distribution.

1.1 Problem Statement and Setup

We consider a set of rankings $\mathcal{T}$, where each $\tau_i \in \mathcal{T}$ has a domain D_{τ_i} and is of size k. The global domain of items is then $D = \bigcup_{\tau_i \in \mathcal{T}} D_{\tau_i}$. We investigate the impact of various choices of k on the query performance in the experiments. Figure 1 shows three example rankings.

$$\tau_1 = [2, 5, 4, 3, 1]$$
$$\tau_2 = [1, 4, 7, 5, 2]$$
$$\tau_3 = [0, 8, 7, 5, 6]$$

Table 1: Example rankings

Rankings are represented as arrays or lists of items; where the left-most position, in the examples written in-line, denotes the top-ranked item. The rank of an item i in a ranking τ is given as $\tau(i)$.

At query time, we are provided with a query ranking q, where $|D_q| = k$ and $D_q \subseteq D$, a distance threshold θ_d, and distance function d. Our objective is to find all rankings that belong to $\mathcal{T}$ and have a distance less than or equals to θ_d, i.e.,

$$\{\tau_i | \tau_i \in \mathcal{T} \wedge d(\tau_i, q) \leq \theta_d\}$$

As mentioned above, rankings can be interpreted as short sets and we can build an inverted index over them, to look up—at query time—those rankings that have at least one item overlapping with the query's items. For the rankings found this way, the distance to the query is evaluated. Considering the example in Figure 1, for a query ranking $q = [8, 7, 0, 6]$, ranking τ_1 does not overlap at all with the query's items, while τ_2 and τ_3 do overlap. The retrieval of overlapping candidates using an inverted index, that keeps for each item a list of pointers to rankings that contain the item, is very efficient [15]. The index is accessed for each of the query's items and, for the found candidate rankings, the distance function is applied with respect to the query and the true results are returned. Note that we assume the distance threshold θ_d to be strictly smaller than the maximum possible distance (i.e., normalized maximum distance equal to 1). Thus the inverted index can find all result rankings as all results need to have at least one overlapping item with the query.

Kendall's Tau is defined as the pairwise disagreement between two permutations of a set, which suggests building an inverted index that is labeled (as keys) by pairs of items. Moreover, two ordered pairs can be combined into a triplet which then also carries information about pairwise disagreement. In this paper, we present how exactly these indices can be used by relating them to LSH schemes, show that they are in fact locality sensitive, and theoretically derive bounds on the number of index accesses needed to reach a specific recall goal.

1.2 Contributions and Outline

In this work, we make the following contributions:

- We first propose ad-hoc similarity search over sets of top-k lists (aka. rankings), considering naive inverted indices over single items, pairs, or triplets. We also derive pruning rules to improve the efficiency of ad-hoc similarity search by eliminating false-positive results with help of a distance bound established in this paper for given distance thresholds.
- We are able to theoretically derive bounds on the expected recall for the parameters of query-driven LSH, and describe how to automatically tune a crucial parameter in the proposed LSH method to make NN-search efficient.
- We compare the performance of the presented LSH schemes to traditional inverted indices and to the Sim-Join method—a competitor based on the popular prefix filtering framework.

A preliminary version of this paper has been published at the WebDB'14 workshop [23]. The remainder of this paper is organized as follows. Section 2 gives a brief overview on the Kendall's Tau distance, LSH, and inverted indices. Section 3 discusses the derivation of a distance bound for the plain inverted index. Section 4 shows the consequences of interpreting rankings as sets of pairs and triplets, which motivates to propose LSH schemes for Kendall's Tau, discussed in Section 5. Section 6 presents the query-driven LSH method for the proposed LSH schemes and develops a way to automated parameter tuning in order to increase the efficiency of similarity search. Section 7 discusses the experimental results. A brief literature study is presented in Section 8. Section 9 concludes the paper.

2. PRELIMILARIES

2.1 Kendall's Tau on Top-k Lists

Complete rankings are considered to be permutations over a fixed domain D. We follow the notation by Fagin et al. [11] and references within. A permutation σ is a bijection from the domain $D = D_\sigma$ onto the set $[n] = \{1, \ldots, n\}$. For a permutation σ, the value $\sigma(i)$ is interpreted as the rank of element i. An element i is said to be ahead of an element j in σ if $\sigma(i) < \sigma(j)$. The Kendall's Tau distance $K(\sigma_1, \sigma_2)$ measures how both rankings differ in terms of concordant and discordant pairs: For a pair $(i, j) \in D \times D$ with $i \neq j$ we let $\bar{K}_{i,j}(\sigma_1, \sigma_2) = 0$ if i and j are in the same order in σ_1 and σ_2 and $\bar{K}_{i,j}(\sigma_1, \sigma_2) = 1$ if they are in reverse order. Then Kendall's Tau is given as $K(\sigma_1, \sigma_2) = \sum_{i,j} \bar{K}_{i,j}(\sigma_1, \sigma_2)$.

In this work, we consider incomplete rankings, called top-k lists in [11]. Formally, a top-k list τ is a bijection from D_τ onto $[k]$. The key point is that individual top-k lists, say τ_1 and τ_2, do not necessarily share the same domain, i.e., $D_{\tau_1} \neq D_{\tau_2}$. Fagin et al. [11] discuss Generalized Kendall's Tau which can be computed over top-k lists and is not a metric. We adopt the generalized Kendall's Tau distance function as defined by Fagin et al. [11]:

DEFINITION 1. *Given two top-k lists τ_1 and τ_2 that correspond to two permutations σ_1 and σ_2 on $D_{\tau_1} \cup D_{\tau_2}$, the generalized Kendall's Tau distance with penalty p, denoted as $K^{(p)}(\tau_1, \tau_2) = \sum_{i,j} \bar{K}_{i,j}(\sigma_1, \sigma_2)$ is defined as follows:*

- *Case 1: If $i, j \in D_{\tau_1} \cap D_{\tau_2}$ and their order is the same in both list then $\bar{K}^{(p)}(\tau_1, \tau_2) = 0$ else $\bar{K}^{(p)}(\tau_1, \tau_2) = 1$.*

- *Case 2: If $i, j \in D_{\tau_1}$ and i or $j \in D_{\tau_2}$, let $i \in D_{\tau_2}$ and $\tau_1(i) < \tau_1(j)$ then $\bar{K}^{(p)}(\tau_1, \tau_2) = 0$ otherwise $\bar{K}^{(p)}(\tau_1, \tau_2) = 1$.*
- *Case 3: If $i \in D_{\tau_1}$ and $j \in D_{\tau_2}$ or vice versa then $\bar{K}^{(p)}(\tau_1, \tau_2) = 1$.*
- *Case 4: If $i, j \in D_{\tau_1}$ and $i, j \notin D_{\tau_2}$ or vice versa then $\bar{K}^{(p)}(\tau_1, \tau_2) = p$.*

Case 4 in Definition 1 associates a penalty p to the distance if both the elements of a pair are absent from one list. In such cases, *it is difficult to predict the order of these missing elements if they appear in the top-k list and, secondly, their absence from one of the top-k list decreases their significance.* Therefore, we consider penalty $p = 0$ for case 4 in Definition 1 in this work. The generalized Kendall's Tau Distance with penalty 0 is denoted as $K^{(0)}$ in this work.

2.2 Locality Sensitive Hashing (LSH)

The key idea behind LSH is to map objects into buckets via hashing, with the key property that similar objects have a higher chance to collide (in the same bucket) than dissimilar ones.

DEFINITION 2. *A Locality Sensitive Hashing scheme is a distribution on a family $\mathcal{H}$ of hash functions operating on a collection of objects $\mathcal{T} \in \mathbb{R}^d$. With a distance function d, distance threshold r, and approximation factor $c > 1$, $h \in \mathcal{H}$ satisfies the following conditions for any two objects $o, q \in \mathcal{T}$*

- *if $d(o, q) \leq r$ then $Pr_{\mathcal{H}}(h(o) = h(q)) \geq P_1$*
- *if $d(o, q) \geq cr$ then $Pr_{\mathcal{H}}(h(o) = h(q)) \leq P_2$*

The hash family $\mathcal{H}$ becomes effective for similarity search when $P_1 > P_2$. To make the difference between P_1 and P_2 large, instead of using one random hash function to map an object into a hash bucket, m random hash functions $h_j \in \mathcal{H}$ are concatenated to create a function family $\mathcal{G}$. Usually, $m < d$. To achieve higher recall in searching, l functions $g_i \in \mathcal{G}$, where $g_i = (h_{1,i}, h_{2,i}, \ldots, h_{m,i})$, $1 \leq i \leq l$, are used to create l hash tables where data points are mapped.

In query processing, the query q is hashed to the l hash tables using functions $g_i \in \mathcal{G}$. Andoni and Indyk [1] explained that LSH solves the *randomized r-NN reporting problem.* All data points that reside in the same bucket with q in any of l hash tables are considered candidates for the query result. Subsequently, the candidates are validated by calculating their distances to the query object.

The probability P_1^m that a data point o is hashed into the same bucket with q is $Pr(g_i(o) = g_i(q))$. This means, the probability of colliding in at least one of the l buckets is $(1 - (1 - P_1^m)^l)$. Hence, the error probability δ, i.e., the probability of having an r-NN of the query that does *not* collide in any of l hash tables is at most $(1 - P_1^m)^l$.

Therefore, based on δ and r the probability that true positive candidates are returned by LSH is given by:

$$1 - \delta \geq 1 - (1 - P_1^m)^l \tag{1}$$

$1 - \delta$ in Equation 1 presents the recall function of the LSH method. In this paper, we propose two hash families for distance function $K^{(0)}$. Unlike the traditional preprocessing of data in LSH, where only l hash tables are populated with data, we propose three different inverted indices in Section 3. We establish the relation between these indices and the hash tables of LSH in Section 5. In Section 6, we further discuss the tuning of the parameter l based on the error probability δ and a biased selection of l hash functions at query time.

2.3 Inverted Index

Rankings can be interpreted as plain sets, ignoring the order among items. Hence, one obvious way to index sets of items is to create a mapping of items to the rankings in which the items are contained in. This resembles the basic inverted index known from information retrieval and also used for querying set-valued attributes [15].

An inverted index consists of two components—a *dictionary* $\mathcal{D}$ of *objects* and the corresponding *posting lists* (aka. index list) that record for each object information about its occurrences in the relation (cf., [31] for an overview and implementation details).

3. INV. INDEX WITH DISTANCE BOUNDS

As discussed earlier, it is possible to find similar rankings by retrieving all rankings from the inverted index that *overlap at least in one item* with the query items. We use such a basic inverted index on rankings, illustrated in Table 2, as a baseline in the experimental evaluation.

For a user-provided query ranking q and a distance threshold θ_d, the **filter and validate** technique works as follows:

- The inverted index is looked up for each element in D_q and a candidate set $\mathcal{C}$ of rankings is built by collecting all distinct rankings seen in the accessed posting lists.
- For all such candidate rankings $\tau \in \mathcal{C}$, the distance function $K^{(0)}(\tau, q)$ is calculated and if $K^{(0)}(\tau, q) \leq \theta_d$ then τ is added to result set $\mathcal{R}$.

Potentially, many of the candidate rankings in $\mathcal{C}$ are so called *false positives*, i.e., rankings that are found when accessing the posting lists but do not belong to $\mathcal{R}$. Each such false positive causes an unnecessary distance function computation. Intuitively, $\tau \in \mathcal{R}$ should be found in at least a certain number of posting lists, depending on the distance threshold θ_d. In this section, we establish a criterion that allows removing some of the false positives by computing the minimal number of posting lists that need to be accessed, for the elements in D_q. This idea is similar to the *prefix filtering* method [12], except that elements in D_q do not follow any *global* ordering.

Lemma 1 determines the minimum overlap that need to exist between two rankings such that they can have a distance less than threshold θ_d. This result is then used to make Proposition 1 which derives the bound on the number of elements in D_q need to be looked up in the inverted index to retrieve all true results.

LEMMA 1. *The* **minimum number of overlapping elements** *between two rankings τ and q to have at most a Kendall's Tau distance θ_d is given by $\mu = (k - \sqrt{\theta_d})$, where $|D_q| = k$.*

Proof: Let μ be the minimum number of overlapping elements between two rankings q and τ. The *minimum* possible distance $K^{(0)}(\tau, q)$ with μ overlapping elements can be found by considering the following constraints: (i) all overlapping elements are in the same order in both q and τ, i.e., $K^{(0)}(\tau, q) = 0$ for Case 1 and (ii) all non-overlapping elements of the ranking appear at the *bottom* of both lists, i.e., $K^{(0)}(\tau, q) = 0$ for Case 2 according to Definition 1. Hence, the minimum $K^{(0)}(\tau, q)$ is $(k - \mu)^2$ due to Case 3, according to Definition 1. Now, solving the equation $(k - \mu)^2 = \theta_d$, we get $\boldsymbol{\mu = \lceil k - \sqrt{\theta_d} \rceil}$. Clearly, all rankings with overlap $n < \mu$ will have $K^{(0)}(\tau, q) > \theta_d$ and can be immediately ignored. k^2 is the maximum distance possible between

$7 \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$5 \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle, \langle\tau_3\rangle$
$4 \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
...

Table 2: Basic Inverted Index

$(4,5) \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
$(5,7) \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$(3,4) \rightarrow \langle\tau_1\rangle$
...

Table 3: Sorted Pairwise Index

$(5,4) \rightarrow \langle\tau_1\rangle$
$(7,5) \rightarrow \langle\tau_2\rangle, \langle\tau_3\rangle$
$(4,5) \rightarrow \langle\tau_2\rangle$
...

Table 4: Unsorted Pairwise Index

$(2,4,5) \rightarrow \langle\tau_1\rangle, \langle\tau_2\rangle$
$(2,5,7) \rightarrow \langle\tau_2\rangle$
$(4,5,7) \rightarrow \langle\tau_2\rangle$
...

Table 5: Triple Index

two top-k rankings and we can normalize the distances (i.e., $\theta = \theta_d/k^2$). Thus, μ can be also expressed as $\lceil k(1-\sqrt{\theta})\rceil$.

From the previous lemma, we can directly derive that *all result rankings must appear in at least μ posting lists*, which leads to the following proposition.

PROPOSITION 1. *For retrieving all rankings in $\mathcal{R}$, it is sufficient to look up the posting lists for only $k-\mu+1$ elements from the query's domain D_q.*

Though r-NN search under the Kendall's Tau distance is feasible using Proposition 1 on the basic inverted index, the perspective of the definition of Kendall's Tau distance opens the discussion of having more advanced index structures that we will discuss in next section.

4. PAIRWISE AND TRIPLE INDEX

Kendall's Tau is defined over discordant pairs between two rankings, which immediately suggests treating a ranking as a set of pairs. We can go further and observe that triplets of items can also provide information about the order among item pairs. Therefore, we also consider an inverted index structure using pairs and triplets as key, respectively. This is feasible as top-k rankings are very small compared to the potentially large global domain. In this section, we present three different ways to represent the rankings and propose three different index structures based on those representations. Following the discussion from the previous section, we will also see that at query time not all pairs or triplets of the query's items need to be considered.

4.1 Rankings as Sets of Pairs

Considering a ranking as a set of pairs allows us to directly compare overlapping pairs between two rankings to determine the Kendall's Tau distance. Representing a ranking τ as a set τ_s^p of sorted pairs leads to the following definition.

DEFINITION 3. *The* **sorted pairwise index** *maps a pair $(i,j) \in \tau_s^p$ to a posting list that holds all rankings (ids) that contain both elements i and j, where τ_s^p represents all pairs of elements that occur in ranking τ, defined as*

$$\tau_s^p = \{(i,j) | (i,j) \subseteq D_\tau \times D_\tau \wedge i < j\}$$

The index structure is called sorted pairwise index because, according to the definition of τ_s^p, each key (i,j) follows $i<j$, without reflecting the order of appearance of i and j in τ. For instance, $\tau_{1s}^p = \{(2,5),(2,4),(2,3),(4,5),(3,5),\ldots\}$ for the ranking τ_1 from Figure 1. Due to the ordering, redundant indexing is avoided. For clarification, Table 3 represents part of the sorted pairwise index for the example rankings given in Section 1.1.

In the τ_s^p representation, in order to compute the Kendall's Tau distance, overlapping pairs between the rankings need to be further verified to check the order of items. Hence, we propose another pair-wise representation of rankings as set of unsorted-pairs τ_u^p, such that the order of item appears in the ranking can be preserved.

DEFINITION 4. *The* **unsorted pairwise index** *maps each pair $(i,j) \in \tau_u^p$ to a posting list that holds all rankings (ids) that contain both elements i and j in same order, where τ_u^p represents all pairs of elements that occur in ranking τ with ranking order, defined as*

$$\tau_u^p = \{(i,j) | (i,j) \subseteq D_\tau \times D_\tau \wedge \tau(i) < \tau(j)\}$$

The index structure is called unsorted pairwise index as, according to definition of τ_u^p, the keys in this index are maintaining the order in which the elements appear in the rankings. For example, $\tau_{1u}^p = \{(2,5),(2,4),(2,3),(5,4),\ldots\}$ for the ranking τ_1 from Figure 1. Thus, a posting list for key (i,j) in the unsorted pairwise index holds all rankings where i occurs before j. Table 4 represents part of the unsorted pairwise index for the example given in Section 1.1. The space[1] and time complexity for building both pairwise indices are $O(|\mathcal{T}|\binom{k}{2})$.

The simple filter and validate approach, introduced earlier on the single-item inverted index, can also be used to process queries on these pairwise index structures: We look up the index for each pair $(i,j) \in q_s^p$ and q_u^p, respectively, to build the candidate set $\mathcal{C}$ and then validate $\mathcal{C}$ to find result set $\mathcal{R}$.

Overall, there are $|q_s^p| = |q_u^p| = \binom{k}{2}$ possible pairs of items derived from query ranking q. However, instead of looking up the pairwise index for all $\binom{k}{2}$ pairs in q, we can derive a bound on the number of pairs that we need to look up in order to be able to find all results; we do that for both proposed index structures. According to Lemma 1, all final results must appear in at least $\binom{\mu}{2}$ pairs from q_s^p and q_u^p, respectively, for threshold θ_d. Hence, we obtain the following proposition.

PROPOSITION 2. *For retrieving all rankings in $\mathcal{R}$, it is sufficient to look up the posting lists for only $\binom{k}{2}-\binom{\mu}{2}+1$ pair elements from q_s^p or q_u^p for the given query q and similarity threshold θ_d.*

4.2 Rankings as Sets of Triplets

Based on the representation of a ranking as a set of triplets, denoted as τ^t for ranking τ, another index structure is proposed in this section.

DEFINITION 5. *The triple index maps a triplet $(a,b,c) \in \tau^t$ to a posting list that holds all rankings (ids) that contain all three elements a,b, and c, where τ^t represents all triplets of elements that occur in ranking τ, with an order defined in*

$$\tau^t = \{(a,b,c) | (a,b,c) \subseteq D_\tau \times D_\tau \times D_\tau \wedge a < b < c\}$$

The keys in this index are ordered and, thus, avoid redundant indexing. For instance, $\tau_1^t = \{(2,4,5),(2,3,4),(3,4,5),\ldots\}$. Table 5 represents a part of the index created from the example given in the introduction. *It is clear that the more elements are concatenated to create keys for an inverted index, the average size of a posting list becomes smaller and rankings mapped into the same posting list have higher probability to be in fact similar.* Hence, a significant number of

[1]Considering uniform distribution of elements over rankings.

false positive candidates can be avoided by using larger keys. *On the other hand, the number of generated keys grows drastically with keys getting larger.* For the triplets index, the total $\binom{k}{3}$ keys are possible to create from a top-k ranking ranking τ, i.e., $|\tau^t| = \binom{k}{3}$. Hence, the space[1] and building time complexity of triple index is $O(|\mathcal{T}|\ \binom{k}{3})$.

Again, the simple filter and validate paradigm can be used in query processing using these keys for the triple index. Similarly to above, we can find a bound on the number of triplets from the query ranking that need to be looked up for retrieving the results $\mathcal{R}$. According to Lemma 1, all final results must appear in at least $\binom{\mu}{3}$ triplets from q^t, for a specific threshold θ_d. Thus, we obtain the following proposition.

PROPOSITION 3. *For retrieving all rankings in $\mathcal{R}$, it is sufficient to look up the posting lists for only $\binom{k}{3} - \binom{\mu}{3} + 1$ pair elements from q^t for query q and specific θ_d.*

In practice, we can retrieve all result candidates by accessing much fewer triplets or pairs than the established bound in this section. This is discussed in Section 6 and verified by the experimental study in Section 7.

5. LSH SCHEMES FOR KENDALL'S TAU

Candidate rankings that are fetched from the above indices are eventually evaluated to reveal the actual results similar to the query. Particularly for the pairwise and the triplet-based approach, potentially very many lookups need to be conducted, depending on the value of k. On the other hand, larger keys render the lookups more "precise". In this section, we try to find a bound on the number of required lookups that still allows achieving a high recall value. How exactly such an approximate querying can be assessed is understood by **relating the discussed indices to LSH schemes.** Specifically, we propose two hash families for Kendall's Tau distance and show that they are in fact locality sensitive. For each of these families, we consider two hashing schemes. We further theoretically evaluate their suitability to efficiently retrieve the query results.

5.1 Hash Family 1

We introduce the first hash family, denoted as $\mathcal{H}_1$, that contains projections with respect to elements of the global domain $D = \cup_{\tau \in \mathcal{T}} D_\tau$.

DEFINITION 6. *$h_i \in \mathcal{H}_1$, with $i \in D$, is defined as*

$$h_i(\tau) = \begin{cases} 1, & i \in D_\tau \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Clearly, the hash functions from $\mathcal{H}_1$ project rankings on the presence and absence of individual elements, e.g., $h_2(\tau_1) = 1$, $h_2(\tau_3) = 0$, considering the example from Section 1.1.

5.1.1 Locality Sensitivity of Hash Family 1

Each $h_i \in \mathcal{H}_1$ maps $\tau, q \in \mathcal{T}$ to $\{0, 1\}$ according to the presence of i in τ and q. In Section 3, we have derived the required minimum overlap μ for a ranking with the query in order to have a chance to satisfy the distance threshold θ_d. Hence, the distance threshold θ_d is approximated here as the Jaccard distance between query and data (c.f., $1 - \mu/(2k-\mu)$). For the notation, throughout this paper, P_1, P_2, c, and r refer directly to Definition 2. Considering r as the approximated Jacquard distance for θ_d, it is clear that the probability $Pr[h_i(q) = h_i(\tau)]$ is the same as the Jaccard similarity $\mu/(2k - \mu)$, if $i \in \{D_q \cup D_\tau\}$. Considering the global domain D, all hash functions h_i, where $i \notin D_q \cup D_\tau$, map q and τ to the same bucket (labeled as '0') as $i \notin D_q, D_\tau$. Let $|\overline{D_q \cup D_\tau}| = \lambda$, then, $|D| = 2k - \mu + \lambda$, and the collision probability is $P_1 = (\mu + \lambda)/|D|$. More similar rankings will have more overlapping elements than μ, which increases the collision probability P_1, that is, more similar rankings have higher chance to be placed in the same bucket.

On the other hand, as long as the approximation factor[2] c is strictly larger than 1, rankings that have distance larger than cr, have strictly less than μ overlapping elements. Hence, we can say $P_2 < P_1$. Thus, *the locality sensitive property holds for $\mathcal{H}_1$.*

5.1.2 Function Families for Hash Family 1

Based on the hash family $\mathcal{H}_1$, we propose two LSH function families $\mathcal{G}_1$ and $\mathcal{G}_2$ in a way such that the implementation of hash tables for these function families can directly use the proposed pairwise and triplets-based index.

Scheme 1 is based on function family $\mathcal{G}_1$ which is created by concatenating two hash functions, defined as follows:

$$\mathcal{G}_1 = \{(h_i, h_j) | (h_i, h_j) \in \mathcal{H}_1 \times \mathcal{H}_1 \text{ and } i < j\}$$

Hence, a function $g \in \mathcal{G}_1$, with $g = (h_i, h_j)$, projects a ranking τ to the space $\{0, 1\}^2$. Considering the example from Table 1, $g_1 = (h_4, h_5) \in \mathcal{G}_1$, as $4 < 5$. Hence, $g_1(\tau_3) = (0, 1)$, $g_1(\tau_1) = g_1(\tau_2) = (1, 1)$. Here, we notice that the bucket labeled as $(1, 1)$ for g_1 contains both τ_2 and τ_3 which is similar with the bucket labeled $(4, 5)$ in the sorted pairwise index illustrated in Table 4. Clearly, the bucket label $(1, 1)$ for a hash function $g = (h_i, h_j)$ is represented by the key (i, j) in the sorted pairwise index. Thus, the sorted pairwise index contains all hash table entries with bucket label $(1, 1)$ from all the hash tables introduced by $\mathcal{G}_1$. Now, looking up the index for l pairs $(i, j) \in \tau_s^p$ means applying l different functions $g \in \mathcal{G}_1$ on τ.

Further, we define **Scheme 2** with function family $\mathcal{G}_2$ by combining three hash functions from $\mathcal{H}_1$, according to the following definition:

$$\mathcal{G}_2 = \{(h_a, h_b, h_c) | (h_a, h_b, h_c) \in \mathcal{H}_1^3 \text{ and } a < b < c\}$$

Here, $g \in \mathcal{G}_2$, where $g = (h_a, h_b, h_c)$, projects a ranking τ to $\{0, 1\}^3$. Similar to the case for $\mathcal{G}_1$, looking up a triplet from a ranking τ means accessing the bucket with label $(1, 1, 1)$ in the hash table of $g = (h_a, h_b, h_c)$, where $a, b, c \in D_\tau$. Therefore, it is clear that the triple index contains only $(1, 1, 1)$ bucket labels from all hash tables induced by $\mathcal{G}_2$. Thus, looking up the index for l triplets $(a, b, c) \in \tau^t$ means applying l different functions $g \in \mathcal{G}_2$ on τ.

The query performance for different values of l is investigated in the experimental evaluation in Section 7, for both function families.

5.2 Hash Family 2

Instead of considering hash functions based on projections on individual elements, we now define a hash family $\mathcal{H}_2$ that contains all projections on $D_\mathcal{P}$, where $D_\mathcal{P} = \{(i, j) | (i, j) \in D \times D \text{ and } i < j\}$, i.e., all sorted ordered pairs over D. Hence, $\mathcal{H}_2 = \{h_{i,j} | (i, j) \in D_\mathcal{P}\}$.

Recall from Section 2 that $K^{(0)}(\tau, q)$ is defined as the number of discordant pairs in the domain $D_\tau \cup D_q$, for a ranking τ and query ranking q. To resemble this definition/behavior

[2]A discussion on the value of c is out of scope for this paper as this work does *not* address the c-approximate nearest neighbor search problem.

via hash functions, we define hash functions $h_{ij} \in \mathcal{H}_2$ that project τ on $\{0,1\}$ as follows:

DEFINITION 7. *$h_{i,j} \in \mathcal{H}_2$, with $(i,j) \in D_{\mathcal{P}}$, is defined as*

$$h_{i,j}(\tau) = \begin{cases} 1, & i,j \in D_\tau \text{ and } \tau(i) < \tau(j) \\ 1, & i \in D_\tau \text{ and } j \notin D_\tau \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Unlike $\mathcal{H}_1$, this hash family is reflecting the order of elements in a ranking with respect to the order of elements in the domain D. For example $h_{1,4}(\tau_1) = 0$, $h_{1,4}(\tau_2) = 1$, considering the example from Section 1.1.

5.2.1 Locality Sensitivity of Hash Family 2

After projecting the ranking on $D_{\mathcal{P}}$, it is represented as a string of $\{0,1\}$. For clarification, such a representation for τ_1 and τ_2 under hash family $\mathcal{H}_2$ is shown in Table 6.

Clearly, the generalized Kendall's Tau distance $K^{(0)}$ between them becomes the *hamming distance* between such a representation of rankings due to $\mathcal{H}_2$. And the probability $Pr[h(q) = h(\tau)]$ is equal to the number of projections on that τ and q agree. From the definition of $\mathcal{H}_2$, two rankings will not be projected into the same bucket if the rankings are projected on those sorted pairs that are responsible for the distance. Hence, we obtain the collision probability $P_1 = 1 - \theta_d/|D_{\mathcal{P}}|$ considering r as θ_d. If the distance between rankings is smaller than θ_d then P_1 increases, i.e., if rankings are more similar then the probability to project those ranking into same bucket is larger. As long as $c>1$, we have $P_1 > P_2$ and thus the property of locality sensitive hashing holds for $\mathcal{H}_2$.

5.2.2 Function Families for Hash Family 2

Based on $\mathcal{H}_2$, we define two function families $\mathcal{G}_3$ and $\mathcal{G}_4$ that use the previously introduced unsorted pairwise index. We propose **Scheme 3** with function family $\mathcal{G}_3$, defined by selecting any hash function over $\mathcal{H}_2$, i.e.,

$$\mathcal{G}_3 = \{h_{i,j} | h_{i,j} \in \mathcal{H}_2\}$$

For a $g \in \mathcal{G}_3$, i.e., $g = \{h_{i,j}\}$, the bucket labels '1' and '0' of g are represented respectively by the key element (i,j) and (j,i) in the unsorted pairwise index. Thus, the unsorted pairwise index holds hash table entries for all hash functions in $\mathcal{H}_2$. Hence, a ranking list for $(a,b) \in \tau_u^p$ from the unsorted pairwise index is the same as the bucket where τ is projected by $g = h_{i,j}$ with $\{i,j\} = \{a,b\}$. Hence, looking up the unsorted pairwise index for l pairs from τ_u^p implies applying l different functions $g \in \mathcal{G}_3$ on the query ranking.

We define **Scheme 4** with function family $\mathcal{G}_4$ by combining two hash functions from $\mathcal{H}_2$, according to the following definition:

$$\mathcal{G}_4 = \{(h_{i,j}, h_{x,y}) | (h_{i,j}, h_{x,y}) \in \mathcal{H}_2 \times \mathcal{H}_2, \\ (i,j),(x,y) \in D_{\mathcal{P}} \text{ and } |\{i,j\} \cap \{x,y\}| < 2\}$$

Hence, function $g \in \mathcal{G}_4$, with $g = (h_{i,j}, h_{x,y})$, projects a ranking τ to $\{0,1\}^2$. From the discussion of Scheme 3, we know how the hash table for $h_{i,j} \in \mathcal{H}_2$ is related to the unsorted pairwise index. Following that discussion, we can easily retrieve the entry of the hash table for $g \in \mathcal{G}_4$ by intersecting the posting lists from the unsorted pairwise index for key-pair used in g . For example, if $g = (h_{i,j}, h_{x,y})$ maps a ranking τ to $(0,1)$, we can retrieve the rankings appearing in the same bucket with τ by looking up the index entry combined by keys $((j,i),(x,y))$ from the unsorted pairwise index—note that here we scan key (j,i) and not (i,j) as $h_{i,j}$ maps τ to 0. Applying l different $g \in \mathcal{G}_4$ functions on the query element relates to looking up l different pair combination from unsorted pairwise index.

–	(2, 5)	(4, 5)	(3, 4)	...
τ_1	1	0	0	...
τ_2	0	1	0	...

Table 6: Projections of rankings under $\mathcal{H}_2$

Again, the impact of l on both function families are studied theoretically in the following section and evaluated in the experiments in Section 7.

6. QUERY-DRIVEN LSH

It is clear that both hash families $\mathcal{H}_1$ and $\mathcal{H}_2$ contain a large number of hash functions and rankings in binary space according to the presence or absence of an element (in case of $\mathcal{H}_1$) or a pair of elements (in case of $\mathcal{H}_2$) in a ranking. Moreover, we consider top-k rankings where $k << |D|$. Hence, $\mu << |D|$ and $\theta_d << |D_{\mathcal{P}}|$. Consequently, the bucket that contains rankings projected on the *absence* of an element or pair of elements (i.e., bucket with label '0') contains much more entries than the bucket that contains rankings projected on *presence* of an element or pair of elements (i.e., bucket with label '1'). This non-uniform distribution of rankings in buckets of hash tables requires to investigate how the selection of hash functions can optimize the number of retrieved candidate rankings.

A prominent intuition is that a larger number of shared elements between the ranking and the query increases the probability of the ranking being similar "enough" with the query. Hence, in practice, we consider those hash functions that project the query ranking only on the elements or pair of elements that are present in the query, to avoid the projection of the query in a larger bucket. This selection strategy shrinks the candidate set and, thus, increases the efficiency of the naïve LSH method by avoiding the validation of a large number of false positives. We mentioned in Section 1 that a specific way of choosing hash functions within the families based on distribution of elements in the query can render the LSH method more efficient than a naïve LSH method. Here, we will show that our proposed strategy also accomplishes the same objective without analyzing the distribution of elements in the query. This variation of the proposed LSH scheme is called query-driven LSH and we will discuss how exactly this biased choice of hash functions changes the collision probability of the LSH schemes.

6.1 Refining the Collision Probability and Tuning of Parameter l

According to query-driven LSH, for $\mathcal{H}_1$, all the hash functions which are selected to create hash keys at query time are projections on the elements from the query ranking. Similarly, for $\mathcal{H}_2$, all the hash functions which are selected to create hash keys at query time are projections on ordered pairs from the query ranking. Hence, the collision probability changes for both proposed hash families.

The collision probability P_1 for $\mathcal{H}_1$ becomes μ/k because any one of the shared elements is drawn from a total of k query elements. Similarly, for $\mathcal{H}_2$, all the pairs that are responsible for the θ_d distance between the ranking and the query are drawn from a total of $\binom{k}{2}$ ordered pairs from the query. But it is not possible to create the discordant pairs between two lists according to Case 2 of Definition 1 from the elements of only one list. This case is responsible for the $(k-x)^2$ distance for x overlapping elements between

two lists. According to Lemma 1, a minimum of μ overlapping elements are required for the distance threshold θ_d. Therefore, the distance $(k-\mu)^2$ is not introduced by the pairs possible to form using only the query and at least a distance of $(\theta_d - (k-\mu)^2)$ is possible to get introduced by the pairs that appears in the query. This scenario is a very optimistic one. As the number of overlapping elements is $x \geq \mu$, there are more than $(\theta_d - (k-\mu)^2)$ discordant pairs that can be derived solely based on the query. Let $E(\mu)$ denotes the expected number of overlapping elements. Then, for hash family $\mathcal{H}_2$, the collision probability P_1 becomes $P_1 = 1 - (\theta_d - (k - E(\mu))^2/\binom{k}{2})$.

Based on the refined P_1 values for both hash families, we discuss here the tuning of parameter l by fixing error probability δ. According to the definition of $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$, and $\mathcal{G}_4$, the parameter m holds the value $2, 3, 1$, and 2 for Scheme 1, Scheme 2, Scheme 3, and Scheme 4 respectively. By substituting P_1 and m in Equation 1, we obtain the recall functions for Scheme 1, Scheme 2, Scheme 3, and Scheme 4, respectively, with the error probability δ, as listed below.

$$1 - \delta \geq 1 - (1 - (\mu/k)^2)^l \quad \text{Scheme 1} \quad (4)$$

$$1 - \delta \geq 1 - \left(1 - (\mu/k)^3\right)^l \quad \text{Scheme 2} \quad (5)$$

$$1 - \delta \geq 1 - \left(1 - \left(1 - \frac{\theta_d - (k - E(\mu))^2}{\binom{k}{2}}\right)\right)^l \quad \text{Scheme 3} \quad (6)$$

$$1 - \delta \geq 1 - \left(1 - \left(1 - \frac{\theta_d - (k - E(\mu))^2}{\binom{k}{2}}\right)^2\right)^l \quad \text{Scheme 4} \quad (7)$$

A comparison among schemes based on Equations 4–7 will not be appropriate as the distance threshold used in Scheme 1 and Scheme 2 is the Jaccard distance represented in terms of μ where μ gives the lower bound of overlapping elements required to meet distance threshold θ_d. Therefore, to render all schemes comparable, we find the recall functions for Scheme 3 and Scheme 4 by expressing the collision probability P_1 for $\mathcal{H}_2$ in terms of μ. $\mathcal{H}_2$ projects rankings on ordered pair elements, as explained in Section 5.2.1. In terms of μ, all the pairs that disagree between two rankings must be drawn from $\binom{k-\mu}{2}$ pairs from the query. So, the probability that a pair of items appears in both rankings is $(1 - \binom{k-\mu}{2}/\binom{k}{2})$. But these pairs have a 50% chance to appear in the same order in both rankings. Hence, for $\mathcal{H}_\in$, the collision probability $P_1 = 0.5(1 - \binom{k-\mu}{2}/\binom{k}{2})$. Hence, Equation 6–7 is reformulated and the recall function for Scheme 3 and Scheme 4 is presented by Equation 8–9 based on μ:

$$1 - \delta \geq 1 - \left(1 - 0.5\left(1 - \frac{\binom{k-\mu}{2}}{\binom{k}{2}}\right)\right)^l \quad (8)$$

$$1 - \delta \geq 1 - \left(1 - 0.25\left(1 - \frac{\binom{k-\mu}{2}}{\binom{k}{2}}\right)^2\right)^l \quad (9)$$

Figure 1 shows the comparison among the four LSH Schemes given by Equations 4, 5, 8, and 9. It presents how the error probability gets lower (i.e., recall increases) as l increases, for threshold $\theta = 0.1$ and $k = 10$. A predefined recall value$(1 - \delta)$ determined by given error probability δ for all proposed schemes can be reached by varying m and l. In our proposed schemes, as we fixed the m value previously, the recall can be assured and tuned by only varying

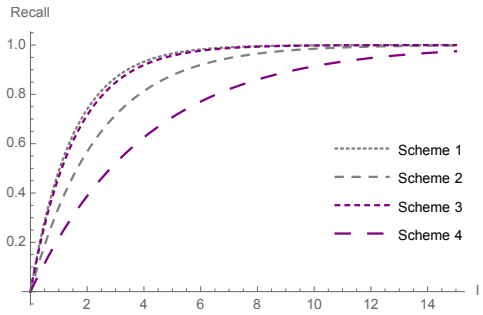


Figure 1: Comparison of recall functions among query-driven LSH schemes; $k = 10$.

	k=30				k=40			
θ	0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4
InvIn+Drop	10	14	17	19	13	18	22	26
Scheme 3	3	4	6	10	3	4	6	8
Scheme 4	4	7	14	33	4	7	14	22

Table 7: Lower bound of the parameter l

the parameter l, that is, the number of hash tables we need to access to retrieve all candidates for similarity search. For instance, Figure 1 shows that Scheme 1 and Scheme 3 can reach 99.99% recall ($\delta = 0.0001$) for $l = 8$ for $\theta = 0.1$ (i.e., $\mu = 7$). This means that 8 hash tables (i.e., 8 entries from both pairwise indices) are enough to retrieve 99.99% of the true positive candidates. We clearly see that this bound is much tighter (to obtain 100% recall) compared to the one established earlier in Proposition 2, discussed in Section 4.1, which would need to look up 25 entries (i.e., three times more). This improvement is consistent for the other schemes, too.

A tighter bound for l can be calculated by solving Equations 6 and 7, as these equation use the exact Kendall's tau distance threshold θ_d and not using a more relaxed threshold μ. Depending on δ, the value of l can be higher than $\binom{k}{2}$. Hence, we consider accessing $min(\binom{k}{2}, l)$ hash tables in the query-driven LSH in order to avoid picking the same hash function multiple times. Table 7 presents a comparison with respect to the number of hash tables (keys in pairwise and triple index) that need to be scanned compared to the InvIn+Drop method ($\delta = 0.01$). We see that our proposed methods scan significantly less entries compared to InvIn+Drop for lower θ. And this difference in performance with respect to the baseline is getting larger as the ranking size k increases, thus, making our schemes more efficient.

In practice, we observe that even 100% recall can be obtained with l tuned as shown in Table 7, for $\delta = 0.01$. We further discuss this in Section 7.1 for top-10 and top-20 lists. Figure 1 shows that the recall functions for Scheme 1 and Scheme 2 give an upper bound for the recall functions Scheme 3 and Scheme 4, respectively. For these characteristics, the parameter l for Scheme 3 and Scheme 4 can also be used for Scheme 1 and Scheme 2, respectively, to achieve a predefined recall value. This is validated by our experimental results. Hence, these tighter bounds of l are used to tune the parameter l in query-driven LSH schemes.

6.2 Effect of the Position of Query Elements

Reflecting again how Kendall's Tau is given in Definition 1, we notice that a non-overlapping element that appears at a top rank is responsible for more discordant pairs than if it would appear at a bottom rank. Hence, the list

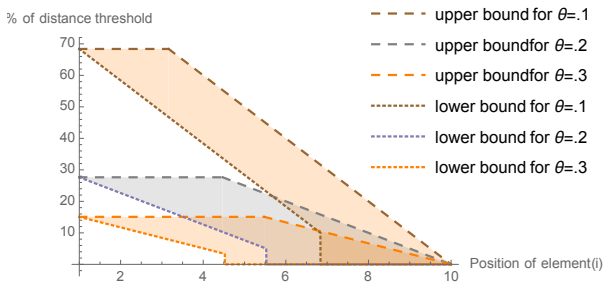


Figure 2: Number of discordant pairs due to different positions (rank) of elements in a top-10 ranking

with more non-overlapping elements in the top is more likely not to pass the similarity threshold. Here we discuss how this characteristic effects the candidate pruning during similarity search. In this paper, we denote the total number of discordant pairs caused by any non-overlapping element that appears in the i^{th} position as $\bar{K}_i^{(0)}$. For any two top-k rankings and a given threshold θ_d, the upper and lower bound of the $\bar{K}_i^{(0)}$ can be calculated as follows:

$$_{up}\bar{K}_i^{(0)} = min(k-i, \mu) \text{ where } \mu = k - \sqrt{\theta_d} \tag{10}$$

$$_{low}\bar{K}_i^{(0)} = \begin{cases} 0, & i \geq \mu \\ \mu - i + 1, & \text{otherwise} \end{cases} \tag{11}$$

$$_{avg}\bar{K}_i^{(0)} = 1/2(_{low}\bar{K}_i^{(0)} +_{up} \bar{K}_i^{(0)}) \tag{12}$$

According to Equations 10 and 11, Figure 2 shows how the rank of a non-overlapping element contributes to the distance, expressed in percentage of θ. It is clear from Figure 2 that, theoretically, for a small distance threshold (such as $\theta = 0.1$), the pruning of false positive candidates can be achieved more effectively for the hash functions that project the rankings on the elements from top positions, as missing elements from very top positions is responsible for more than 50% discordant pairs of the total Kendall's Tau distance.

Therefore, selecting hash functions that project on the elements from the topmost positions have a higher probability to retrieve true positive candidates. In Figure 2, we also see that the lower and the upper bound of contribution to the distance for the i^{th} non-overlapping element decreases as the threshold increases. Hence, selecting hash functions by giving preference to top-ranked elements becomes comparatively less effective in the candidate pruning for larger distance thresholds. As our proposed LSH schemes use pairs or triplets from the query rather than using unique elements for projecting rankings, focusing on only top elements will have a higher chance to retrieve duplicate rankings as candidates. Hence, l functions for query-driven LSH schemes are chosen according to Algorithm 1. It ensures that functions are chosen by selecting hash functions that project the query emphasizing *not only* top elements of the query.

If we replace line 5 in Algorithm 1 with "$T = \{g_i | h_j \in g_i \wedge E(h_j) \in q\}$ ", the algorithm will return l functions that *only* emphasize on top elements from the query. In Section 7.2, a comparison between random selections of hash functions over position-influenced selections is discussed with experimental results for real-world datasets.

Algorithm 1: Selecting l functions for the query-driven LSH schemes

Require: query $q = \{e_1, e_2, \ldots, e_k\}$; $q(e_i)$ = position of e_i
weight of e_i, $W(e_i) =_{avg} \bar{K}_i^{(0)} / \sum_{e_j \in q} {}_{avg}\bar{K}_i^{(0)}$;
$\mathcal{H} = \{h_1, h_2, \ldots\}$; $W(h_i) = \sum_{e_j \in E(h_i)} W(e_j)$;
$\mathcal{G} = \{g_1, g_2, \ldots\}$; $W(g_i) = \sum_{h_j \in g_i} W(h_j)$
$E(h_x)$ = elements used in h_x for projecting q, $h_x \in \mathcal{H}$;

1: $S := \emptyset$
2: $S := argmax_{g_i}\{W(g_i) | h_j \in g_i \wedge E(h_j) \in q\}$
3: **while** $|S| \leq l$ **do**
4: $\quad \mathcal{G} := \mathcal{G} - S$
5: $\quad T := argmin_{g_i}\{\sum_{e_a, e_b \in \bigcup_{h_x \in g_i}} |q(e_a) - q(e_b)|$ where $h_j \in g_i \wedge E(h_j) \in q\}$
6: $\quad S := S \cup argmax_{g_i}\{W(g_i) | g_i \in T\}$
7: **end while**
8: **return** S

7. EXPERIMENTS

We have implemented the index structures described above in Java 1.6 and run the experiments using an Intel Xeon CPU @ 2.67GHz machine, running Linux kernel 3.2.60, with 264GB main memory. The index structures are kept entirely in memory.

To evaluate the querying performance in terms of query response time (here, **wallclock time**), **number of retrieved candidates**, and **recall** (i.e., fraction of results found), we use two different datasets.

Yago Entity Rankings: This dataset contains 25,000 top-k rankings that have been mined from the Yago knowledge base, as described in [16]. Essentially, it contains entity rankings like the top-10 tallest buildings in the world or the largest countries by number of inhabitants.

NYT: This dataset contains search engine result rankings, created using 1 million keyword queries, randomly selected out from a published query log of a large US Internet provider. These queries are executed against the New York Times document corpus [27] using a standard tf*idf scoring model from the information retrieval literature.

The datasets are different in nature, specifically considering the frequency in which items appear across different rankings. In the Yago dataset which holds real-world entities, each entity occurs in a few rankings only, while the NYT dataset comprises many popular documents that appear in many query-result rankings.

In addition to the baseline approach described in Section 3, we also consider a competitor based on the work by Wang et al. [28]. The concept of prefix filtering was used in their work, where an adaptive framework for improving the performance of executing joins between two sets is proposed. They also propose an algorithm for similarity search, called SimJoin Query, on top of their adaptive framework that is coined Adapt-Join. We implemented their proposed data structure, called *Delta Inverted Index*, and use *SimJoin* as the main competitor for our proposed schemes.

Overall, a comparative study on the following approaches are presented:

- The filter and validate technique on the simple inverted index denoted as **InvIn**.
- The filter and validate technique on the simple inverted index combined with dropping selectively larger posting lists from consideration using the distance bound given in Section 3, denoted as **InvIn+Drop**.

	NYT (MB)		Yago (MB)	
	k=10	k=20	k=10	k=20
Simple Inverted Index	14.99	22.08	15.99	30.18
Sorted Pairwise Index	64.81	267.45	107.0	437.59
Unsorted Pairwise Index	45.45	188.25	77.51	319.06
Triple Index	127.0	776.18	254.2	1042

Table 8: Comparison of size among indices

	NYT (sec)		Yago (sec)	
	k=10	k=20	k=10	k=20
Simple Inverted Index	0.081	0.114	0.075	0.095
Sorted Pairwise Index	0.342	1.732	0.368	1.886
Unsorted Pairwise Index	0.307	1.434	0.409	1.816
Triple Index	1.250	12.09	1.434	28.69

Table 9: Comparison of building time among indices

- The presented LSH **Scheme 1**, using the sorted pairwise index.
- The presented LSH **Scheme 2**, using the triplets index.
- The presented LSH **Scheme 3**, i.e., the unsorted pairwise index.
- The presented LSH **Scheme 4**, i.e., using the unsorted pairwise index.
- The competitor **SimJoin**.
- We have additionally implemented a method **LinearScan** that is simply a full linear scan over all rankings for comparison. We do not report on it in the plots for readability, as **LinearScan** *is at least 34 times slower than* **InvIn**.

Table 8 and Table 9 present a comparison of the building time and the index size among different indices.

7.1 Performance Analysis among Query-Driven LSH Schemes and Competitors

The runtime performance is measured in terms of average runtime for 1000 queries while varying the *normalized* distance threshold θ (given by $\theta_d = k^2 \times \theta$). For all four proposed LSH schemes, Table 10 presents the value of the parameter l calculated manually such that 100% recall is achieved in similarity search. This table also presents the value of l which is tuned automatically according to the theoretical discussion in Section 6.1 with error probability $\delta = 0.01$, confirming recall value 99%. From Table 10, we can see that the automatically tuned parameter l leads almost all of the time to manually tuned l for 100% recall. For the cases where the manually tuned l is larger than the theoretically established one, the actual recall for this value of l is given within parentheses.

The baseline approach InvIn retrieves total of 22,755 and 37,071 candidates and takes on average 0.05ms and 0.15ms to response, respectively for top-10 and top-20 Yago rankings. InvIn retrieves a total of 60,627 and 74,128 candidates and takes on average 0.15ms and 0.35ms to response, respectively for top-10 and top-20 NYT rankings. **The performance of InvIn is more than 3 times slower in Yago and 2 times slower for NYT compared to our proposed schemes.** From Figure 3, we see that fewer candidates are retrieved in all proposed schemes from the pairwise indices or the triplets index, compared to the baseline and the InvIn+Drop method, in case of the Yago dataset. This performance improvement stems from the tight bound of parameter l, resulting in less keys that need to be accessed, as well as from shorter posting lists due to the query-driven selection of hash tables. Since recall is tuned to 100%, retrieving fewer candidates means evaluating fewer false-positive candidates, which reflects the characteristic of the LSH technique that true-positive candidates are more likely to be hashed into the same bucket. The characteristic of retrieving less false-positive candidates than the baseline and InvIn+Drop methods is consistently observed throughout the datasets—except for Scheme 1 and Scheme 3 with parameter $\theta = 0.1$ for the NYT dataset (cf., Figure 4).

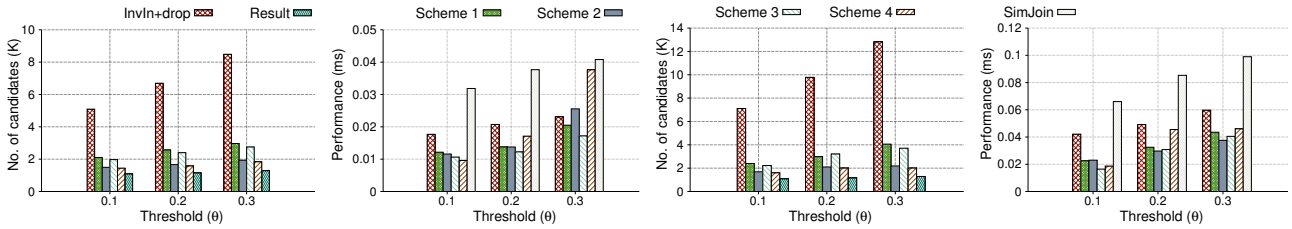
In Figure 3 and Figure 4, we see that retrieving less false-positive candidates for all the proposed schemes directly influences the runtime performance for both datasets. For all proposed schemes, as the threshold increases, more hash tables need to be accessed, but the runtime performance remains almost proportionate with the number of retrieved elements, for all approaches that we compare in this work, for both datasets. The only exception we observe for Scheme 4 with $\theta = 0.3$, due to the computation time for finding intersection of posting lists from the key pairs used in Scheme 4 as discussed in Section 5.2. Figure 3 and Figure 4 show that all the proposed schemes perform better than the competitor SimJoin. SimJoin finds the best prefix scheme for each query and applies Adapt-Join using a delta inverted index to find potential candidates. From the experimental results, for both datasets, we can conclude that the 1-prefix scheme performs best most of the time, as the InvIn+Drop method performs better than the competitor. More precisely, the InvIn+Drop scheme uses the 1-prefix method and ensures best performance by dropping the longest index entries. Unlike this, in the SimJoin method, the index entries for the elements are globally sorted by the inverted document frequency and, thus, dropping posting lists according to the adaptive best prefix scheme does not ensure the dropping of larger ones. For the Yago dataset, we can see that all the schemes perform better than InvIn+Drop (except Scheme 4 for top-10 rankings with $\theta = 0.3$). For the NYT dataset only Scheme 2 and Scheme 4 is winning over InvIn+Drop, except Scheme 4 with $\theta = 0.3$. One reason for this is that the NYT dataset contains more skewed data which is further explained by a recall analysis among the schemes using experimental results summarized in Table 11.

In Figures 3 and 4, we notice that Scheme 2 and Scheme 4 consistently retrieve fewer candidates than Scheme 1 and Scheme 3, respectively. Figure 5 shows that the precision of Scheme 2 and Scheme 4 is always higher than for Scheme 1 and Scheme 3. This result implies that the probability of retrieving candidates that belong to $\mathcal{R}$ in Scheme 2 and Scheme 4 is higher than in Scheme 1 and Scheme 3, respectively. This is in line with the LSH property that the probability of finding similar ranking is higher when more hash functions are used to create hash keys for LSH schemes.

Comparing the individual rows in Table 11, we see that the percentage of recall reached by Scheme 2 and Scheme 3 is less than Scheme 1 for same value of l. This observation validates our theoretical upper bound among the schemes shown in Figure 1 for different l. Putting it in another way, Scheme 3 is less likely to find a true-positive result than Scheme 1 for the same value of l. In addition, comparing the columns of Table 11, we observe that the recall increases as l increases, which is in line with the LSH theory. We also understand the characteristic of the datasets by analyzing the recall. Comparing the rows of Table 11, for all the schemes, with the same threshold θ and l value, we can see that the recall for the NYT dataset is always larger or equal to the recall for the Yago dataset. This reflects that the elements of the

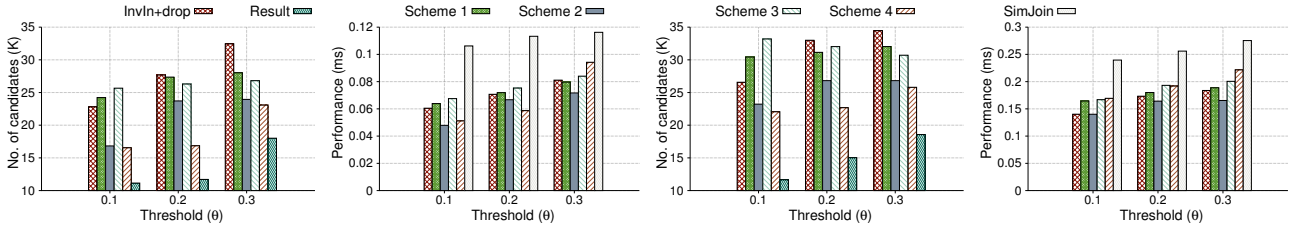
k	θ	Data	Scheme 1		Scheme 2		Scheme 3		Scheme 4	
			Theoretical	Manual	Theoretical	Manual	Theoretical	Manual	Theoretical	Manual
10	0.1	NYT	3	1	4	2	3	2	4	2
		Yago	3	3	4	3	3	3	4	3
	0.2	NYT	5	4	8	6	5	5	8	6
		Yago	5	4	8	6	5	5	8	6
	0.3	NYT	7	5	18	9	7	7	18	13
		Yago	7	6	18	11	7	7	18	15
20	0.1	NYT	3	3	3	3	3	3	3	3
		Yago	3	3	3	3	3	3	3	3
	0.2	NYT	6	6	8	8	6	6	8	8
		Yago	6(99.9)	7	8(99.8)	9	6(99.9)	7	8(99.8)	11
	0.3	NYT	7	7	10	10	7	7	10	10
		Yago	7(99.9)	8	16	12	7(99.9)	9	16	12

Table 10: Comparison of tuning factor l for 100% recall



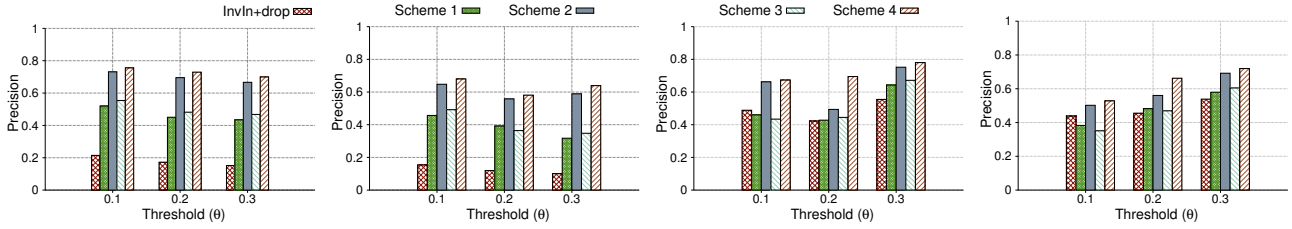
(a) $k = 10$ (b) $k = 10$ (c) $k = 20$ (d) $k = 20$

Figure 3: Comparative study of query processing for varying θ (Yago)



(a) $k = 10$ (b) $k = 10$ (c) $k = 20$ (d) $k = 20$

Figure 4: Comparative study of query processing for varying θ (NYT)



(a) $k = 10$, Yago (b) $k = 20$, Yago (c) $k = 10$, NYT (d) $k = 20$, NYT

Figure 5: Comparative study of precision for varying θ

NYT dataset are more skewed than in the Yago dataset.

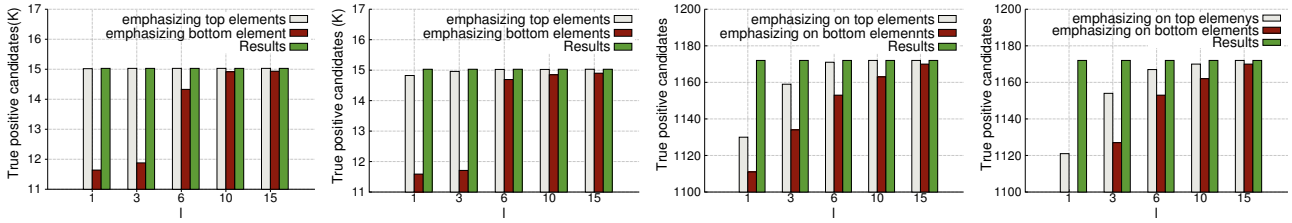
Figure 5 presents the comparison by precisions of all four schemes. It also reflects that the NYT dataset is more skewed than Yago. Due to skewed distribution of elements in the NYT dataset, the precision of all schemes for NYT is almost the same as the precision of InvIn+Drop, whereas precision of all four schemes for Yago data is much higher than the precision of InvIn+Drop.

7.2 Performance Analysis among Different At-Query–Time Selection Methods

Figure 6 reports on the effect of the position of elements on selection of hash function for Scheme 1 and Scheme 3 using $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. It shows that both schemes retrieve more true-positive candidates when hash functions are selected emphasizing on pairs or triplets from top elements of the query rather than the bottom elements, for both datasets. For the same reason, we also see that the difference between retrieved true-positive candidates for focusing on the very top and the very bottom elements is larger in NYT rather due to skewed distribution than Yago. Therefore, 100% recall is also achieved faster by pruning false-positive candidates shown in Figure 6. This property remains consistent to all other schemes.

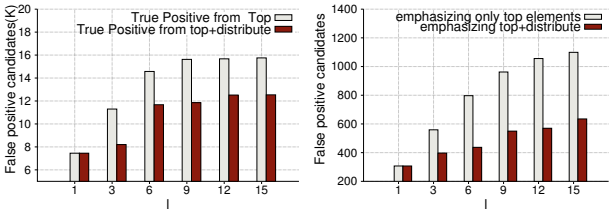
	$\theta = 0.1$				$\theta = 0.2$				$\theta = 0.3$				
	l=1	l=3	l=6	l=10	l=1	l=3	l=6	l=10	l=1	l=3	l=6	l=10	l=15
Scheme 1 for NYT	100	100	100	100	99.9	100	100	100	99.2	99.9	100	100	100
Scheme 2 for NYT	99.9	100	100	100	99.8	99.9	100	100	85.4	85.6	100	100	100
Scheme 3 for NYT	99.7	100	100	100	98.6	99.5	99.9	100	97.5	99.2	99.8	100	100
Scheme 4 for NYT	99.8	100	100	100	98.5	99.7	99.8	100	83.5	85.0	86.2	100	100
Scheme 1 for Yago	99.1	100	100	100	96.4	98.8	99.9	100	92.0	96.8	99.2	99.6	100
Scheme 2 for Yago	99.1	100	100	100	95.6	98.0	99.6	100	90.5	95.2	98.4	99.6	100
Scheme 3 for Yago	99.0	100	100	100	95.6	98.4	99.5	99.8	90.8	96.3	98.9	99.5	100
Scheme 4 for Yago	98.8	100	100	100	94.7	97.7	98.9	100	89.1	94.9	98.2	100	100

Table 11: Comparison of achieved recall in percent for $k = 20$



(a) Scheme 1, NYT (b) Scheme 3, NYT (c) Scheme 1, Yago (d) Scheme 3, Yago

Figure 6: Comparative study of selection of hash functions for top-20 query with $\theta = 0.2$



(a) NYT (b) Yago

Figure 7: Effect of selection method for Scheme 2, $k = 20$ and $\theta = 0.2$

Figure 7 shows that for the same value of l, less false-positive candidates are retrieved using the selection method according to Algorithm 1 than the selection method according to the same algorithm when replacing Line 5 (i.e., to make the selection emphasizing only the top elements) for both datasets. This effect also remains consistent with other schemes. This result has direct influence in efficiency of similarity search as retrieving more false positive results need more validation time to come up with result set $\mathcal{R}$. Efficiency suffers more as the ranking length increases because validation time is proportionate with the ranking length. Considering Scheme 2, we need at least 8 entries to look up for 100% results (given in Table 10) for top-20 list and scanning 8 entries need to validate 3772 more false positive candidates while hash function are selected only emphasizing top elements (see Figure 7). This difference becomes larger for more skewed dataset and it grows as l grows due to generation of more duplicate candidates.

7.3 Lessons Learned

We can summarize the main lessons that we can learn from the above experimental results as follows.

1. The value of the automatically tuned parameter l for 99% recall matches almost precisely the actual value of l that is needed to retrieve all results.
2. Comparing the efficiency of similarity search among all proposed schemes, Scheme 2 performs best for skewed data (e.g., NYT) whereas Scheme 3 performs best for non-skewed dataset.
3. There is a tradeoff between the space needed to store the index structure for the proposed schemes and the efficiency during similarity search. Though Scheme 3 does not perform best for skewed dataset, it uses less space to store the hash tables (i.e., the unsorted pair-wise index) compared to the space needed to store the hash tables for Scheme 2 (i.e., the triplets index).
4. All the schemes outperform the adaptive prefix-filtering method for similarity search.

8. RELATED WORK

There is ample work on computing relatedness between ranked lists of items, such as to mine correlations or anti-correlations between lists ranked by different attributes. Work on rank aggregation [10, 26] aims at synthesizing a representative ranking that minimizes the distances to the given rankings, for a given input set of rankings. Arguably, the two most prominent similarity measures are Kendall's tau and Spearman's foot rule. Fagin et al. [11] study comparing incomplete top-k lists, i.e., lists capturing a subset of a global set of items, rendering the lists incomplete in nature. Scenarios that motivates our work, like similarity search over favorite/preference rankings, lists are naturally incomplete, capturing, e.g., only the top-10 movies of all times. Helmer and Moerkotte [15] present a study on indexing set-valued attributes as they appear for instance in object-oriented databases. Retrieval is done based on the query's items; the result is a set of candidate rankings, for which the distance function can be computed. For metric spaces, data-agnostic structures for indexing objects are known, like the M-tree by Ciaccia et al. [7, 30]; but Kendall's Tau over incomplete list is not a metric. For Spearman's Footrule distance, which remains a metric even for top-k lists, Milchevski et al. [22] use a mixture of metric space indexing and inverted indices for

processing ad-hoc similarity queries. Wang et al. [28] propose an adaptive framework for similarity joins and search over set-valued attributes, based on prefix filtering. This framework can be applied in the filter and validate technique on the naïve inverted index discussed in this work.

The key idea behind Locality Sensitive Hashing (LSH) [1, 9, 8, 13, 24] is the use of locality preserving hash functions that map, with high probability, close objects to the same hash value (i.e., hash bucket). Different parameters of locality preserving functions together with the number of hash function used, render LSH a parametric approach. Studies concerning LSH parameter tuning [9, 3] have been performed providing insight into LSH parameter tuning for optimal performance. LSH can be extended for non-metric distance using reference object has explained in [2]. In literature, search methods using variants of the LSH technique can be found for all-pairs similarity search [25] or K-Nearest-Neighbor (KNN) search [21, 12]. Our objective is to retrieve all rankings within a given threshold for a non-metric distance function—the generalized Kendall's Tau distance. Jégou et al. [19] present a query-adaptive LSH method where only the highest relevant hash functions to the query are selected based on the E_8 lattice formation of them, to solve the nearest-neighbor problem. Gao et al. [12] present data sensitive hashing (DSH) based on the concept of LSH, where they bound the NN search by top-k objects.They also propose methods to select hash functions to increase the efficiency of search incorporating the distribution of query data.

9. CONCLUSION

In this paper, we presented an efficient approach to processing similarity queries over top-k lists under the generalized Kendall's Tau distance. We proposed four different LSH schemes using two different hash function families, reflecting different ways to realize and understand pair-based and triplet-based indices. Key contribution of this work is the in-depth analysis of the ability of the proposed methods to determine high-recall results with few look-ups on the index. We have derived query-driven formulations of the expected recall and presented bounds that allow an automated tuning of the number of hash tables required to achieve a predefined recall. We implemented the described approaches and reported on the results of a comprehensive performance evaluation, using two real-world datasets. This study confirmed the insights obtained through theoretic analysis and further demonstrated the superiority of the approaches over plain inverted indices and SimJoin, the state-of-the-art from literature.

10. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *FOCS*, 2006.

[2] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. *ICDE*, 2008.

[3] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. *WWW*, 2005.

[4] J. L. Bentley. K-d trees for semidynamic point sets. *Symp. on Comp. Geometry*, 1990.

[5] B. Carterette. On rank correlation and the distance between rankings. *SIGIR*, 2009.

[6] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. *CIKM*, 2009.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *VLDB*, 1997.

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Symp. on Comp. Geometry*, 2004.

[9] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. *CIKM*, 2008.

[10] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. *WWW*, 2001.

[11] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Math.*, 17(1), 2003.

[12] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi. DSH: data sensitive hashing for high-dimensional k-nnsearch. *SIGMOD*, 2014.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *VLDB*, 1999.

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD*, 1984.

[15] S. Helmer and G. Moerkotte. A performance study of four index structures for set-valued attributes of low cardinality. *VLDB J.*, 12(3), 2003.

[16] E. Ilieva, S. Michel, and A. Stupar. The essence of knowledge (bases) through entity rankings. *CIKM*, 2013.

[17] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *STOC*, 1998.

[18] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM TOIS*, 20(4).

[19] H. Jegou, L. Amsaleg, C. Schmid, and P. Gros. Query adaptative locality sensitive hashing. In *ICASSP* 2008.

[20] R. Kumar and S. Vassilvitskii. Generalized distances between rankings. *WWW*, 2010.

[21] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. *VLDB*, 2007.

[22] E. Milchevski, A. Anand, and S. Michel. The sweet spot between inverted indices and metric-space indexing for top-k-list similarity search. *EDBT*, 2015.

[23] K. Pal and S. Michel. An LSH index for computing kendall's tau over top-k lists. *CoRR*, abs/1409.0651, 2014.

[24] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[25] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 5(5), 2012.

[26] F. Schalekamp and A. van Zuylen. Rank aggregation: Together we're strong. *ALENEX*, 2009.

[27] The New York Times Annotated Corpus. `http://corpus.nytimes.com`.

[28] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. *SIGMOD*, 2012.

[29] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *VLDB*, 1998.

[30] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with m-trees. *VLDB J.*, 7(4), 1998.

[31] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.